Failures of Hardware and Software Co-Design Projects in History

B. Bordihn, C. Elmali, D. Fruhner, C. Göhner, A. Heller, A. Lottis, L. Mewes, M. Raschke, A. Strauss

Fachhochschule Dortmund, Fachbereich Informatik,
Emil-Figge-Str. 42, 44227 Dortmund

Abstract

This paper deals with known failures in the field of Hardware and Software Co-Design. In detail three examples are illustrated; the dead loss of the Navy Ship USS Yorktown, the self-destruction of the Ariane 5 rocket and the life-threatening failure of the Therac-25. In addition the connection between Hard- and Software is explained.Therefore the prevention of failures in this area will be determined.
Finally ways of testing the reliability of a system will be described. It also addresses a strategy that shows how the problems between hardware and software can be recognized and solved strategically.

I. INTRODUCTION

To turn an idea into reality, planning is essential. Especially in respect of industry projects, systems engineering becomes more important due to the increase of the complexity of projects. Additionally, the field of Hardware and Software Co-Design gains influence. It is often considered to be the subject of computer science and it represents the interface of Hardware and Software components of the planning. (1)

The expenditure of a project depends on several factors. The more complex a project is, the more failures could probably occur. Accordingly, project planning provides the opportunity to detect and to remove failures early during the project. Still a successful completed planning is not a warranty for an error–free project. Failures often lead to immense costs or even to the loss of human lives. Beside pressure of time and costs, reasons for undetected errors are often poor consultation of the particular project fields. Hereafter the following examples show the lack of communication regarding the field of Hardware and Software Co-Design. It also illustrates how mistakes could have been prevented or which experiences have been gained by these project failures. (1) (2)

The paper shows examples for faulty project plannings in computer science, where the error occurred after the project close–out. Afterwards conclusions for the reasons of failure are drawn and existing techniques to make a system reliable are presented.

To keep in mind “I am not discouraged, because every wrong attempt discarded is another step forward“.
Thomas Alva Edison (1847-1931) (9)

II. CASE STUDIES

USS Yorktown

In 1996 the Navy Ship, type *Guided Missile Cruiser* was upgraded with the *Smart Ship Concept* to pare down personnel. The *Smart Ship Concept* linked the whole ship within one network and it was controllable with 27 integrated terminals. The system which was used for operations was Windows NT 4.0.

On 21st September 1997 the USS Yorktown was practicing a maneuver. During this maneuver a security program was incorrectly announcing a closed valve to be open. Surprisingly this kind of error was not unusual. At most time the error was corrected by a system officer directly in the database itself. These manual corrections to the system were quoted handwritten as well so that the progress could be reversed if necessary at a later time. In case of this open valve, the officer changed the value in the database to zero. That has never been done before.

Without any testing of the entered value by the system software, the value was committed to a division-function. This caused *Division by Zero*-Errors. Because of the occurring exceptions, the temporary memory was quickly filled. Reaching the maximum of available memory, an overflow of data occurred. Thus the following data were directly written in the main memory of the driving system. Unfortunately these undesired data overwrote important system data. So it entailed a totally breakdown of the network and the complete driving

system. As a result the ship was floating in open water incapable of acting. After at least three hours it was towed to the harbor in Norfolk.

This example reveals the dimension of the outcome by errors which were done throughout the planning stages. The emerging question of guilt could clearly be assigned to the management. Apparently the management underrated the development of the *Smart Ship Concept*. As a consequence, they hired too little developers for such an immense project. So the planning, the phases of testing and the implementation were inadequate. Also the testing or rather the validation was not properly accomplished because of time pressure. As well later there was only slightly preventive maintenance of the *Smart Ship Concept*.

For the breakdown certain errors can be identified. There was neither a check of the input nor a restriction for a specific range to the value. Additionally the memory entries were unprotected. The critical data of the system was not protected from overwriting and a possible buffer overflow was not taken in consideration. The avoidance of buffer overflows and the protection of data could have been done with methods from Hardware- or Software-Management. Apparently the aspect of hardware was not considered very well for the project planning. Moreover there was no separation between different hardware components. Otherwise such a trivial error would not have incapacitated the whole ship. Apparently a better maintenance could have avoided the appearance of these errors. (3) (4)

Ariane 5

The European Space Agency was founded in the 1970. This space agency started the Ariane program to develop satellite launcher systems. On 24th December 1979, the first Ariane rocket was launched into space. The 4th version of the Ariane launcher was capable of transporting up to 4.9 tons of payload into space. During time, the demands on the Ariane launcher steadily rose. Therefore, starting from scratch a new Ariane was developed, Ariane 5. This type of rocket should have been able to carry more payload than the versions that have been designed before. It took 10 years for the development with costs of 5.5 billion euro until the day of the maiden flight. (5)

On 4thJune 1996, the maiden flight of the Ariane 5 launcher was supposed to take place from the Guiana Space Centre Kourou in French Guiana. For this mission, four cluster satellites should be transported into space. These were planned to be used to determine the magnetosphere of earth. Finally, after some delay caused by the weather, the Ariane 5rocket was launched at 9.34 am. For the first 37 seconds of flight, Ariane 5 behaved as expected but then the trajectory changed extremely. Caused by this the rocket started the mechanism for self-destruction at about 4000 meters of heights. The financial loss amounted to 290 million euro and the Ariane programme had a delay to about one year. (8)

The disaster was caused by a failure in the Internal Reference System (SRI). It was responsible for determining the position of the rocket. This also included the measurement of the horizontal speed. The SRI was almost completely taken over from the predecessor, the Ariane 4. Due to the higher speed and the faster acceleration an error occurred in the SRI by a conversion from a 64-bit floating point number to a 16-bit signed integer number. This arithmetic overflow occurred because of the excessive horizontal velocity of Ariane 5. This was five times higher than the horizontal velocity of the Ariane 4. As a result to this error the SRI sent diagnostic information instead of information about the position to the On Board Computer (OBC). The OBC took this information as flight data and calibrated a deviation in the trajectory. Finally to compensate the deviation, the OBC transmitted a signal to the jet propulsions. By the instruction, the deviated trajectory adjusted to an angle of more than 20 degrees. The Ariane 5 was not able to withstand the existing aerodynamic forces and started to fall apart. The rocket began the self-destruction modus and exploded. (5) (8)

This disaster occurred because the software, which was designed for the hardware of Ariane 4, was almost completely taken over and integrated into the new hardware of Ariane 5. In the processing of the horizontal velocity, a conversation error occurred. This error would not have occurred with Ariane 4 because of the much lower horizontal velocity.

The failure is the result of incorrect requirements analysis and an inadequate verification and validation phase to the compatibility of the two different hardware systems. The *Failure Modes and Effects Analysis* could have been a method to expose the failure and therefore to prevent the disaster. (5)

Therac-25

The Therac-25 was an electron-linear accelerator being used in radiotherapy to treat particularly cancer. The machine has been able to generate electron beams for a superficial treatment as well as X-rays for the treatment of deeper tissue. From 1985 to 1987 many cases of insufficient radiation doses occurred. There were also six accidents being caused by an excessive amount of radiation, three ended fatal.

The Therac-25 had two preceding models which were developed by AECL (Atomic Energy of Canada Limited) with the French company CGR (Compagnie General Radiographique). These were called Therac-6 and Therac-20. Like these two preceding models, the Therac-25 also used a PDP-11 Computer as a controller, which was very popular back then. However, the Therac-6 and Therac-20 are based on preceding models of CGR which were not controlled by software.

After its detachment from CGR, AECL decided to construct the Therac-25 from the start with a complete software control. Safety functions like monitoring the beams and limiting the dose were only implemented in the software.

Another important role for proper operation of the Therac-25 played a turntable under the electron beam. According to the desired mode the turntable could be set in three positions:

- Electron mode [E]
- Photon mode (X-ray mode) [X]
- Field Light Position (used for correct positioning of the patient)

If the photon mode is activated by software and the turntable is set in electron mode or in the field light position at the same time, the patient will receive an excessive overdose of radiation (approx. 100 times of the intended dose).

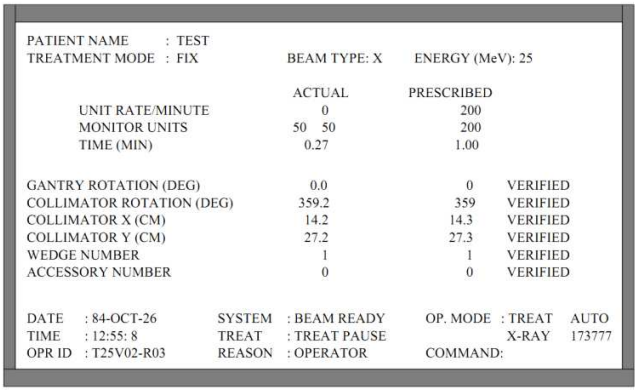

Figure 1 Operator Interface (6)

Figure 1 shows the terminal where the operator entered the complete treatment prescription. The input data were applied when the curser was in the COMMAND field in the right bottom corner.

Problems occurred when entering the beam type (X) followed by *curser down* to the COMMAND field and then *curser up* to change the beam type (E) and once again *curser down* to the COMMAND field within eight seconds. The corrections of the operator were not taken over by the system.

One reason was that a flag was set during alignment of the bending magnets which took eight seconds. Also the synchronization between keyboard handler and treatment monitor task did not work properly so that a possible race condition[1] was set up.

Another overdose occurred as the beam was in the field light position. This happened because a variable in the software was increased during machine setup. It was an 8-Bit variable and could contain a maximum value of 255 decimal. A value of zero represented a correct turntable position. That means every 256th pass through an overflow occurred and no further check of the turntable position was done and a correct setup of the device was assumed by the system.

The correct position of the turntable was only surveyed by the software. There was no second test through hardware backups. Besides, AECL used the software module of the Therac-6 and the Therac-20 for constructing the Therac-25 without testing once more the correctness. By this, errors of the preceding models were transferred.

The software has not been sufficiently shaped and thus, errors have emerged in the whole SW system. Due to *Probabilistic Risk Assessment* all errors would have been noticed instantly or throughout an entire simulation. In the course of the development of the SW, the single components of the system have not been tested sufficiently. The harmonious interaction of the SW and the HW has not been ensured either. (6) (7)

III. Strategies of Failure Analysis

When having a project where hardware and software work as one system, failures of hardware and software components may occur.

Throughout time, hardware ages and so the system loses its stability and may create predictable or random failures. In case of hardware failures, software can be influenced.
Software will become more stable as bugs have been found and fixed. Unfortunately changes that have been done can increase the rate of failure if the new version isn't sufficiently tested. A combined software/hardware system contains different sources of failure.

There are several ways of testing the reliability of a system. Three of these will be describe briefly within the next paragraph.

One method is called *Failure Modes and Effects Analysis* (FMEA). It divides the system into subsystems and even into individual components. All of these will be separately analyzed and a likelihood of failure will be drawn of each component. Thus it is possible to assess the risk and reliability of the whole system. It follows the principle of the *bottom-up* method. In contrast to this the *Fault Tree Analysis* (FTA) uses the *top-down* method. FTA systematically searches for causes of an error in the system. A possible malfunction of this will logically be divided and the components of the system that might cause this error will be evaluated. *Probabilistic Risk Assessment* (PRA) is a method which is used to quantify the risk of failure. To do so, different techniques for fault/event modeling are incorporated for example master logic diagrams, event sequence diagrams or fault trees. These build a probabilistic framework to guide decision-making throughout the design process. (4)

Besides these methods there are graphical evaluation tools. One of these tools is the *Functional Failure Identification and*

[1] A race condition is anomalous behavior caused by the unexpected dependence on the relative timing of events. In other words, a programmer incorrectly assumed that a particular event would always happen before another. (10)

Propagation (FFIP) that incorporates a configuration model, a functional system model, a behavior model, a system behavior simulator, and a function failure logic reasoned. FFIP concatenates hierarchical system models with behavioral simulation and qualitative reasoning. To use FFIP properly the proposed software-hardware system design methodology follows five main steps:

1. To analize the system, the component and functional model need to be created
2. Determin for each component the unique failed states to a certain input flow and output flows
3. The components which have an effect on the propagation of failures need to be identified
4. Construct scenarios of initiating failures
5. Multiple scenarios need to be compared to identify effective changes to the design

(4)

IV. CONCLUSION

Almost any machine can be seen as a system or rather as a combination of single components. The interaction of these components makes up the desired functionality.

The examples that were introduced consisted of hardware and software components. It is clear to see that the correct functionality is only given if every single part of the whole system works properly.

Within each scenario one component of the system was inaccurate and therefore the complete system as well. Through completed and detailed requirement analysis from an early stage, the important requirements can be tested individually. Furthermore it is possible to create a simulation of system fragments to reduce errors in software. These malfunctions could have been found and fix throughout extensive tests or simulations.

Especially if the failure of a system causes a danger for human lives the reliability and the precise functionality should be first priority no matter that it needs more time to produce the system and it is costlier.

V. LIST OF LITERATURE

1. **T. Mahr, R. Gessler.***Hardware-Software-Codesign: Software-Entwicklung für flexibler Mikroprozessor-FPGA-Hochleistungssysteme.* Wiesbaden : Vieweg+Teuber, 2007. ISBN: 3834800481.

2. **Schaumont, P. R.** *A Practical Introduction to Hardware/Software Codesign.* blacksburg VA : Springer, 2010. ISBN: 1441959998.

3. **Lotz, R.** USS Yorktown. *Seminar: Berühmt berüchtigte Softwarefehler* . Universität Koblenz-Landau, 2003.

4. **I. Tumer, C. Smidts.** Integrated Design-Stage Failure Analysis of Software-Driven Hardware Systems. *IEEE TRANSACTIONS ON COMPUTERS.* Dezember 17, 2010, p. 14. Preprint, DOI: 10.1109/TC.2010.245.

5. **Weyand, C.** Ariane 5-Luftfahrt . *Seminar: Berühmt berüchtigte Softwarefehler* . Universität Koblenz-Landau, 2003.

6. **Leveson, N.** *SafeWare: system safety and computers.* University of Michigan : Addison-Wesley, 1995. ISBN 0201119722.

7. **Pfeifer, Martin.** Therac-25. *Seminar: Berühmt berüchtigte Softwarefehler* . Universität Koblenz-Landau, 2003.

8. **Tomoo, Matsubara.** Ariane 5-Luftfahrt. *Paper: Ariane 5: Who Dunnit?*Matsubara Consulting 1-9-6 Fujimigaoka, Japan, 2007.

9. **worldofquotes.com**. Inspirational Quotes, Quotations, and Sayings, http://www.worldofquotes.com/topic/Inspirational/6/index.html, Last visit: 21st June 2011

10. **FreeBSD Developers' Handbook.** Race Conditions, http://www.freebsd.org/doc/en/books/developers-handbook/secure-race-conditions.html, Last visit: 22nd June 2011